An Integrated Framework for Learning Fundamentals in Computer Networks

Qian Liu
Math & Computer Science Department
Rhode Island College
Providence, RI 02908
qliu@ric.edu

***Abstract*—Fundamentals in Computer Networks are essential to one's deep understanding of network internals. Among those networking fundamentals, TCP (Transmission Control Protocol) related mechanisms and techniques are important elements not only because it is widely used in Internet, but also because its design principles have significant effect on the design of other network protocols. In general, network simulators and/or animations are used in Computer Networks courses to illustrate TCP along with other fundamentals, however, those approaches do not provide ways for students to delve into TCP core techniques. In this paper, we introduce an integrated framework for students to explore, practice, and learn TCP fundamentals in detail. It can interact with students, direct them in each step during their learning process, and allow them to implement protocol functionalities or to adjust existing ones as needed. It creates a personalized learning environment in which students can learn networking fundamentals at their own pace.**

Keywords—Computer Networks, TCP, Network Simulator, Personalized Learning

I. Introduction

Networking fundamentals are building blocks of modern networks, and a strong foundation of those basics is very important to student success and to prepare them for advanced skills. Among those fundamentals, TCP (Transmission Control Protocol) related techniques and mechanisms are core elements not only because numerous applications in today's Internet rely on it, but also because its design principles have significant effect on the design of other networks. In networking education, TCP fundamentals, especially error control, flow control, congestion control, and connection management, are essential topics in a typical, introductory networking course although there are often differences in the teaching order or covered depth [1].

TCP guarantees reliable data delivery on top of an unreliable transmission service by using error control. Several techniques, such as packet sequence number, acknowledgement, and retransmission timer are combined to provide reliability, to calculate network measurements such as latency and round-trip-time, and to detect problems such as packet loss, out-of-order, or duplicated packets. Equipped with flow control and congestion control mechanisms, TCP can also prevent a fast sender from overwhelming a slow receiver in order to reduce the need for data retransmission, thus forming a complete framework that not only provides reliability, but also controls end-to-end connections and nodes' behaviors. Many other networks use TCP design principles as basis to build their own transport protocols. For instance, similar techniques in TCP error control, flow control, and congestion control are used in high performance networks such as InfiniBand [2], RoCE (RDMA over Converged Ethernet) [3], and are also used to implement mechanisms such as reliable multicast [4-5]. Therefore, these fundamentals are not only essential parts in a networking course, but also important to prepare students for advanced skills.

In this article, we introduce an integrated framework that can help students comprehend networking fundamentals, especially TCP error control, flow control, congestion control, connection management, and state transition. Compared to the existing simulators and learning models, our framework has the following four advantages:

- The framework creates a personalized learning environment for students. They have freedom to configure the framework as needed and learn things at their own pace.
- The framework interacts with students to help them learn essential techniques and mechanisms. It guides students in each step and provides feedback and hints when they make mistakes, thus building a self-directed and self-explanatory learning environment.
- The framework is based on the *socket* model and allows students to implement protocol functionalities or to adjust existing ones in it. Students have opportunities to consider more protocol design and implementation details, to do more hands-on practice and to gain better program design skills.
- The framework can be running in two modes: GUI (Graphic User Interface) mode and command line mode, which have different implementations and design principles. Students can work in either or both modes based on their needs.

The rest of this paper is organized as follows. Section II discusses related work. Section III introduces the design and deployment of our framework. Section IV discusses the effectiveness of the framework and student results. Section V draws the conclusion.

II. Related work

Network simulators are usually introduced into networking courses to illustrate protocol behaviors and to compare network performance in various scenarios without requiring dedicated hardware. Some [6-9] are using simulation as animation to visualize network algorithms or fundamentals to students. One big concern about those approaches is that they usually illustrate network concepts in an ideal scenario. In actual network circumstances, however, network events are not well-organized, and do not occur in the same order as in the simulations. Therefore, those animations are not able to equip students with the ability of analyzing abstract concepts on their own [10], and students' understanding of networking fundamentals may stay at the theoretical level.

Simulators like OMNeT++ [11] and NS-3 [12] are open-source tools that provide abstractions for network functionalities such as sending, receiving, queuing, forwarding, etc. They provide several built-in simulation examples so that students in introductory networking courses can simply run those examples and then observe the simulation output to experience network behaviors without the need to understand simulator internals. Students can also adjust protocol attributes, such as MSS (Maximum Segment Size), congestion control algorithms (Tahoe, Reno, etc.), or packet bit error rate in those built-in examples to compare performance among various configurations. For instance, it will be convenient to compare the changes in *cwnd* (congestion window) among different congestion control algorithms in those simulators. However, if students need to design experiments with specific traffic patterns, network topologies, or to collect statistics that are not currently recorded by existing experiments, they must implement these features by following the internal libraries in OMNeT++ or NS-3. This requires one to have a thorough understanding on the simulator internals, for instance, the representation of network elements (end node, switch, router), the logic and procedure used in the simulated sending and receiving, along with many simulator-private structures and interfaces which are not associated with network fundamentals nor with general *socket* library, therefore, it will create a steep learning curve that goes beyond the course requirements and puts extra burden on students in introductory networking courses. In general, these simulators are widely used in new network model and/or protocol development [13-16].

GNS3 (Graphical Network Simulator 3) [17] is a network emulator that virtualizes commercial network devices (especially Cisco routers) and allows one to configure, test, and troubleshoot networks in a virtual environment mirroring the actual network topology without requiring dedicated hardware. GNS3 has no relationships with NS-3 discussed previously though it is "Graphical NS3". One can drag and drop network elements (hosts, switches, and routers) in GNS3 GUI to build and manage networks easily. Students who are preparing for CCNA (Cisco Certified Network Associate) certificate will gain the required in-depth knowledge [18] by using GNS3. It is also a great help for students to go over subnet fundamentals and management especially routing and switching topics because all network elements need to be configured manually in GNS3. For instance, one needs to set up IP and gateway addresses for all (virtual) hosts in GNS3, and to configure all routers by using actual router config commands to activate network connectivity among subnets (or to deny specific traffic between hosts). In addition, GNS3 allows one to capture traffic flows on every link in its virtual network, and in the real world however, it is not possible to see how data are transferred in external networks. This feature gives students opportunity to examine and study how packets are transferred among different subnets and how packet headers are altered from source to destination. However, GNS3 does not generate application layer traffic such as HTTP, FTP, nor does it provide ways for students to adjust protocol behavior or traffic patterns so that students cannot use it to explore details such as TCP error control, flow control, or congestion control.

Packet Tracer simulator [19] is designed by Cisco, it has similar drag-and-drop GUI and provides similar functionalities like GNS3. One big difference between GNS3 and Packet Tracer is that Packet Tracer supports more application layer protocols such as HTTP, FTP, SMTP, etc. Besides, it visualizes traffic flows on every link in animation so that users can follow them step by step to learn protocol behaviors. However, as a simulator that implements protocol behaviors internally and only animates the procedure of data communications, it conceals too much of layer details and their relations to upper layers [20]. Although one can keep track of recorded packets and make connections among the packet sequence numbers and ACK numbers to follow the workflow in TCP data transfer, students cannot introduce any user-defined error models to learn how TCP recovers data in a lossy environment to guarantee reliability. In addition, as a "packet visualizer" that only records packet details, Packet Tracer does not interact with students to test their understanding, nor does it simulate essential mechanisms such as flow control or congestion control, therefore, it is not appropriate to study and investigate those TCP essentials in it.

OPNET simulator [21] (OPNET IT Guru) provides a virtual network environment for modeling, analyzing, comparing, and predicting performance of IT infrastructures. Like GNS3 and Packet Tracer, one can drag-and-drop network elements in OPNET GUI and adjust traffic attributes such as traffic type (FTP, HTTP), repeatability, start/stop time, and data size in it. In addition, OPNET makes it easier to collect a great deal of network statistics, ranging from global statistics such as throughput and latency, to protocol level statistics such as *cwnd*. However, it does not illustrate traffic flows, nor does it provide ways to define specific network scenarios. Students cannot use it to investigate protocol details such as how error control handles various errors in data transfer to ensure reliability.

Besides network simulators, several learning models [20, 22-26] have been proposed to help students comprehend networking essentials in different ways while having programming requirements in their learning processes. However, those models either arrange students to work in a specific framework with pre-defined, model-private interfaces, or arrange students to deal with lower layer (Link Layer) implementations or driver implementations, which primarily deal with OS kernel internals such as driver callbacks, system calls and interruptions. These will put extra burden on students in introductory networking courses, and they cannot fully focus on the learning of network fundamentals. In our framework, students are not limited in any specific context, and they have more opportunities to examine protocol/mechanism design details. In addition, students are engaged in the design of our framework and in the control of its behaviors. They can specify customized scenarios and enable/disable specific functionalities to follow network variations step by step and to learn things at their own pace.

III. DESIGN AND DEPLOYMENT

This section discusses the design details of our learning framework and its deployment in introductory networking classes. Our learning framework builds a Programmable, Interactive Environment (PIE) for practicing and investigating networking fundamentals, especially TCP essentials such as error control, flow control, congestion control, connection management, and state transitions. It consists of three modules: demonstration, interaction, and implementation. The demonstration module illustrates the details of TCP error control by using actual network traffic. It allows students to go beyond the predicted scenarios in common simulators, which typically conceal too much of the underlying techniques and do not provide ways for students to investigate error control techniques. PIE allows students to introduce different error models step by step to investigate how TCP detects errors and recovers data in various scenarios so that students can learn those underlying techniques at their own pace, thus creating a personalized learning environment.

The interaction module focuses on packet details, especially the sequence number, ACK number, and packet flags (PSH, ACK, SYN, etc.), which are crucial elements in TCP data transfer. This module is used to test students' understanding of how TCP relies on those header fields to detect errors and perform retransmissions. During interactions, students must provide the correct values of those header fields at each step in a complete TCP communication: from connection setup to connection close, and the module will provide feedback/hints in case there is any error in student response. In addition, because the state transition topic is usually discussed theoretically without any experiment or practice in networking classes, this module integrates TCP states in each step to help students gain in-depth knowledge of those abstract concepts.

The implementation module focuses on flow control and congestion control mechanisms, which are not fully implemented (the "TBD" component in Figure 1 which will be discussed next). Students should design and implement them in PIE based on those mechanism principles. In this module, students must: 1) make sure that the sender will not overflow the receiver's limited receive window; 2) record and adjust the *cwnd* appropriately according to different network events.

PIE can be running in two modes: GUI mode and command line (cmd) mode. The architecture of the GUI-PIE is depicted in Figure 1 (the internal components will be discussed in the following subsections). GUI-PIE is built on top of JDK (Java Development Kit) and uses the *Java Socket* to generate actual network traffic. Therefore, unlike existing learning models that arrange students to program in specific frameworks or follow specific interfaces, no extra burden is introduced to students when they implement new functionalities or adjust existing ones in PIE. One can run the GUI-PIE in either C/S mode (on two hosts, one host is the server, sending side, and the other is client, receiving side) or standalone mode. The GUI-PIE does not rely on any specific OS features, and is an OS independent, portable tool for PCs (Windows, Linux, and Mac).

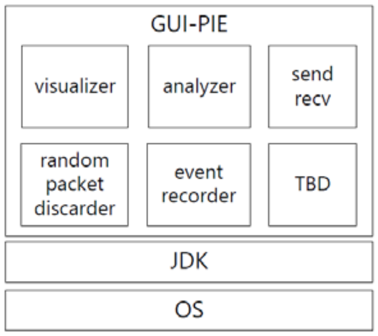


Figure 1: The Architecture of GUI-PIE

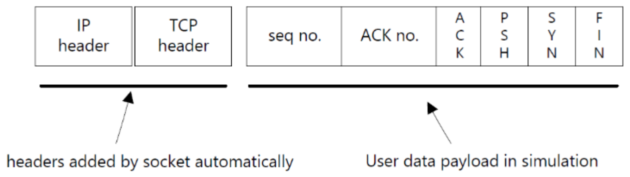
In our framework, the GUI-PIE is separated from the cmd-PIE, which is running on Linux terminal only. Like GUI-PIE, the cmd-PIE can also be running in C/S mode or standalone mode. Unlike GUI-PIE, the cmd-PIE is written entirely in C programming language and uses the *POSIX socket*. The reason we provide the cmd-PIE is that students can realize the differences among various *socket* interfaces, especially the differences in *recv* operation between the *POSIX socket* and the *Java socket and stream classes* from application designers' perspective. Students can work in one mode or both to learn network essentials depending on their needs.

A. Demonstration Module

This module focuses on the TCP error control mechanism, especially the acknowledgement and retransmission techniques in it. We do not explicitly illustrate checksum technique used in error control because the transmission errors (detected by checksum) can be covered by the error model (discussed next) we introduced in PIE. The demonstration module uses detailed illustrations to help students obtain a deep understanding of how packets are delivered reliably in lossy environments, for instance, how does receiver detect packet loss, how does sender realize which packet it sent previously has problem, and when a retransmission is necessary.

Students first specify the total size of the sending data and the size of MTU (Max Transmission Unit), then the sender (*send/recv component* in Figure 1) starts sending data packets to the receiver. The *event recorder component* (shown in Figure 1) records all network events, such as a reception of a packet (data/ACK) and all error cases (discussed next) on both sides. Then the *visualizer component* displays those event logs in the GUI-PIE (in cmd-PIE, they will be displayed as live traffic logs). The *send/recv component* is built by using TCP *socket*, so that the PIE, which resides in the application layer, will not observe any error situations, for instance, packet loss or RTO (Retransmission TimeOut). In order to provide students with error models in which they can investigate how TCP reacts to various network scenarios to guarantee reliability, we introduce the following two mechanisms:

*1) **Explcite ACK**:* after receiving a data packet in PIE, an explicit ACK packet should be constructed and sent to acknowledge the reception of a data packet. It should convey the following simulated information: sequence number, ACK number, and necessary flags, as shown in Figure 2, which is also the data packet format used in PIE.


IP header
TCP header
seq no.
ACK no.
ACK
PSH
SYN
FIN
headers added by socket automatically
User data payload in simulation
Figure 2: Packet Format in PIE

*2) **Random packet discarder component*** (as shown in Figure 1)*:* it is introduced to simulate packet loss, depending on a configurable "loss rate" parameter. That is, after receiving a data packet, the receiver runs the *random packet discarder* to decide whether the current packet should be discarded "manually" as if it was not received previously. On the sender side, after receiving an ACK, it runs the discarder to decide whether to discard the explicit ACK manually. Therefore, unlike the existing simulators/animations that generate predicted traffic patterns, the *random packet discarder* in PIE will generate various error scenarios for students to investigate TCP error control techniques in detail. In addition, the discarder can also be configured to build predicted error scenarios. Students can specify a list of packets in the discarder to only discard those packets, thus building a predicted environment for beginners to learn as needed.

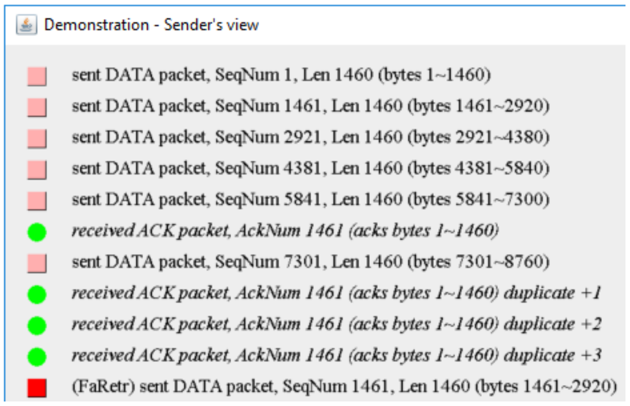
The reasons we use the combination of TCP *socket* and a *random packet discarder*, instead of implementing those features by using UDP, are:

1) students will have control on the introduced error models, and they can simply adjust the value of the "loss rate" attribute to compare and examine various situations;

2) it will be easier to calculate the goodput measurement since the number of delivered packets will be known for sure;

3) this simulation model is extensible and will match students' learning curve if built on TCP. For instance, when students start with the basic concepts in data transfer, the packet discarder can be disabled so that students can get familiar with the overall procedure in the ideal situation; then, the discarder can be introduced to receiver side only so that students can focus on how sender detects packet loss and deals with retransmission based on the retransmission timer or fast retransmission technique; finally, the discarder can be introduced to sender to simulate lost ACK scenarios. This step-by-step process creates a personalized learning environment in which students can learn essential error control techniques at their own pace.

As stated before, all network events, including the "simulated" packet loss, duplicated ACKs, and retransmissions, will be recorded and visualized. Figure 3, for instance, displays events recorded on sender side in a network environment in which the *discarder* is introduced to receiver only. Each event has relevant description and is self-explained. Note that the *discarder* can be introduced to both sides and packets will be discarded randomly, therefore, each time the demonstration module runs in a "new" lossy environment and will generate a new set of events accordingly.


Demonstration - Sender's view

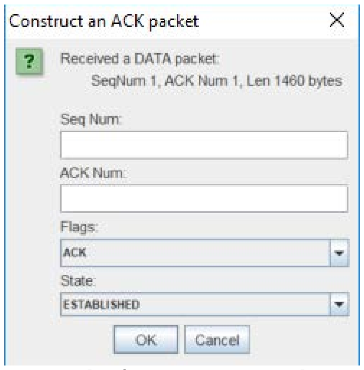
🟥 sent DATA packet, SeqNum 1, Len 1460 (bytes 1~1460)
🟥 sent DATA packet, SeqNum 1461, Len 1460 (bytes 1461~2920)
🟥 sent DATA packet, SeqNum 2921, Len 1460 (bytes 2921~4380)
🟥 sent DATA packet, SeqNum 4381, Len 1460 (bytes 4381~5840)
🟥 sent DATA packet, SeqNum 5841, Len 1460 (bytes 5841~7300)
● *received ACK packet, AckNum 1461 (acks bytes 1~1460)*
🟥 sent DATA packet, SeqNum 7301, Len 1460 (bytes 7301~8760)
● *received ACK packet, AckNum 1461 (acks bytes 1~1460) duplicate +1*
● *received ACK packet, AckNum 1461 (acks bytes 1~1460) duplicate +2*
● *received ACK packet, AckNum 1461 (acks bytes 1~1460) duplicate +3*
🟥 (FaRetr) sent DATA packet, SeqNum 1461, Len 1460 (bytes 1461~2920)

Figure 3: an Example of Recorded Events in the Demonstration Module

It seems that the demonstration module is similar to the packet sniffing tools such as Wireshark analyzer, which captures packet payload and packet header fields such as sequence numbers, packet lengths, and packet types. However, in the real world, it is not easy to let network discard a specific packet in TCP to customize a lossy environment, therefore, tools like Wireshark will not illustrate the gaps in sequence numbers and/or duplicated ACKs. But in the demonstration module, students can easily specify different lossy environments and go through those events to practice the error control techniques such as how receiver detects packet loss, how sender becomes aware of a problematic packet, and when a retransmission is necessary.

B. Interaction Module

This module provides an event-driven interaction for students to learn and practice the mechanisms used in TCP error control to detect error situations such as lost packets, out-of-order packets, duplicated packets, etc. It also provides students with practice in TCP state transition. The interaction module simulates the entire data transfer procedure (from connection setup to connection close) and records all network events, like the demonstration module. Then it uses those events one at a time to interact with students and to test their understanding of how packet header fields are used to acknowledge data reception and to detect errors. First, students choose one side (data packet sending side or receiving side) to begin the interaction. Because different sides have different events, students can get involved in every detail of error control. In each step, students respond to the current event and construct a packet with appropriate values for the packet sequence number, the ACK number, and the packet flag (SYN, ACK, FIN, etc.) in order to move to the next event. Figure 4 illustrates the situation when one host receives a data packet with sequence number 1, ACK number 1, and payload length 1460 bytes. Students should construct an explicit ACK packet with the correct ACK number, sequence number, and choose the appropriate packet flag (ACK) to acknowledge the reception. Then, it moves to the next network event, as shown in Figure 5, the host receives a data packet with sequence number 2921, students can then deduce that a packet may get lost due to the gap in the sequence numbers, then students should provide the appropriate ACK number for the current event.


Construct an ACK packet

Received a DATA packet:
SeqNum 1, ACK Num 1, Len 1460 bytes

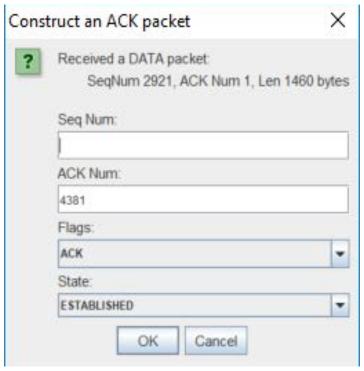
Seq Num:

ACK Num:

Flags: ACK

State: ESTABLISHED

Figure 4: an Example of Constructing an explicit ACK packet


Construct an ACK packet

Received a DATA packet:
SeqNum 2921, ACK Num 1, Len 1460 bytes

Seq Num:

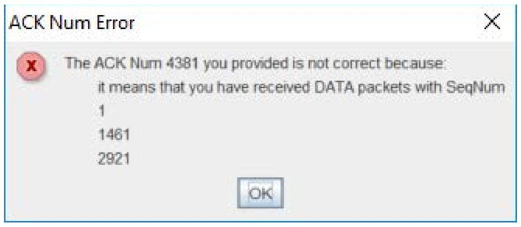
ACK Num: 4381

Flags: ACK

State: ESTABLISHED

Figure 5: an Example of Constructing an explicit ACK packet when gap exists

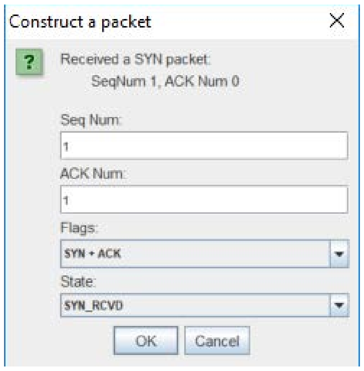
The interaction module provides a self-directive and self-explanatory interface. If any value students provide is not correct, the module reports error information and hints on how to fix it. For instance, if one mistakenly provides ACK number 4381 in the step shown in Figure 5, then the interactive module will report this unaccepted ACK error, in Figure 6, and go back to the previous event (Figure 5), waiting for the correct input values to move to the next event.


ACK Num Error

The ACK Num 4381 you provided is not correct because:
it means that you have received DATA packets with SeqNum
1
1461
2921

Figure 6: Hints for Students if any Value is not correct

The *random packet discarder* is also used in this interaction module. Based on their learning needs and individual progress, students can introduce the discarder on either side step by step to investigate and practice the variations in protocol behaviors because each time PIE runs, it generates a different set of network events. Another way to use the discarder, as discussed previously, is that students can specify a list of discarded packets (discussed in the demonstration module) to build a predictable environment that only discards "pre-defined" packets, thus creating a personalized learning environment. Initially, the discarder is disabled so that no packets get dropped. In this "ideal" scenario, beginners can go over the basic work flow between data packets and ACK packets. Then, the discarder can be introduced on the receiver side only to simulate data packet loss, so that gaps between data packets will appear (for instance, Figure 4 and Figure 5). In this case, students, if working on receiver side, need to acknowledge data packets with "appropriate" ACK numbers. If they are working on sender side, they need to deal with events such as duplicated ACKs and retransmission scenarios. Next, the discarder can be introduced to the sender side to simulate ACK loss scenarios, which will generate more variations. Therefore, instead of observing protocol behaviors, students have opportunities to delve into the details of error control systematically in this module, and they can proceed from simple cases to complex scenarios in accordance with the structure of their knowledge. This helps students obtain a deep understanding of how TCP relies on packet header information and timer to guarantee reliability.


Construct a packet

Received a SYN packet:
SeqNum 1, ACK Num 0

Seq Num: 1

ACK Num: 1

Flags: SYN + ACK

State: SYN_RCVD

Figure 7: an Example of TCP State Transition

In addition, the interaction module integrates TCP state transition procedure, which is usually discussed theoretically in classes without any experiment or practice, so that students are generally not aware of how important it is to TCP communication. Similar transition ideas are applied in other networks, for instance in the RDMA QP (Queue Pair) transition [2]. In each of the interaction steps, especially in the connection setup and connection close, students not only need to provide appropriate values for sequence number, ACK number, and flag, but also need to change the current state, if necessary, to the next appropriate one. For instance, upon the reception of a SYN packet, as shown in Figure 7, one should construct a SYN+ACK packet to accept the connection request and move local state to SYN_RCVD.

C. *Implementation Module*

Students should implement flow control and congestion control mechanisms in this module. In general, these two topics are discussed without hands-on practice in classes, usually diagrams or animations are used to illustrate how they work. In simulators like OMNeT++ or OPNET, *rwnd* (receive window) and *cwnd* (congestion window) statistics are collected automatically so that users can compare how these windows change over time in various configurations. However, those tools only visualize the window changes and do not illustrate the internal principles. It will be better for students to implement these mechanisms in a simple way to understand how to adjust those windows in response to various scenarios.

In the “TBD” component (Figure 1), part of the sending and receiving structure is implemented. Students should continue to use *socket* and the packet format shown in Figure 2 (a new filed “*window*” should be added if it is flow control implementation) to implement flow control and congestion control separately.

In the flow control implementation, sender has knowledge of the initial *rwnd* from a pre-defined, configurable variable “*window size*” (N bytes), and it keeps sending several packets with random size (1 ~ N-1 bytes) of data (students can choose to always send fixed length of data) as long as the window size on the sender side is not 0. If it is the cmd-PIE framework, a simple-TLV (Type-Length-Value) protocol we modified for this framework is used. That is, before sending a data packet with size M bytes (M < N), sender should send a (control) packet first that contains the information of M to indicate the size of the data packet sender goes to send next, so that the receiver (implemented in C program) can call *recv* function with appropriate buffer size to hold the next data packet in order to preserve packet boundary for easy management.

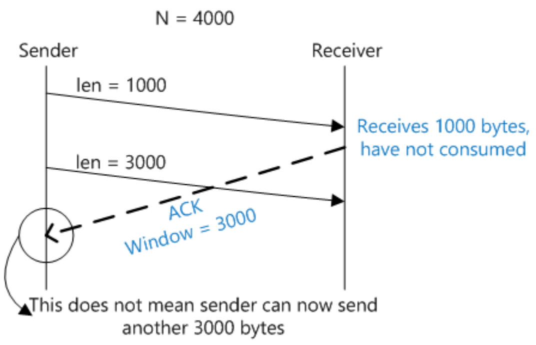

N = 4000
Sender
Receiver
len = 1000
Receives 1000 bytes, have not consumed
len = 3000
ACK
Window = 3000
This does not mean sender can now send another 3000 bytes

Figure 8: an *rwnd* Scenario

On data packet receiver side, it randomly decides whether to consume zero, one, or X data packets (X is less than the number of packets received but not consumed), and then sends an explicit ACK packet with its current window size integrated. For instance, in the scenario shown in Figure 8, students must make sure that sender should not send any data packets upon reception of the explicit ACK.

The partially finished TBD component is implemented in single thread mode. We encourage students to implement flow control in multi-thread mode (for instance in Figure 9) to handle synchronization in multi-thread environment, in which more situations should be considered. This will help students fully understand the underlying flow control mechanism and gain better programming skills. In flow control implementation, the random packet discarder is not used.

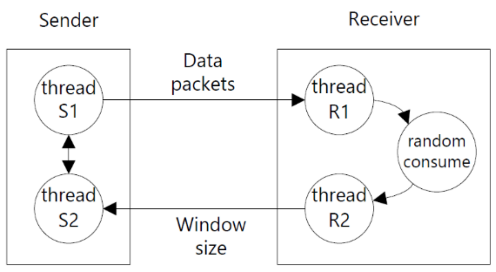


Figure 9: one Possible Flow Control Model in Multi-thread Mode

In congestion control implementation, the sliding window size discussed in the demonstration module should start from 1 (initial slow start window size) MSS, and it should be adjusted appropriately based on different network events according to congestion control algorithm students implement. We suggest students to use *random packet discarder* (with low drop rate) on the receiver side only to discard data packets randomly. Sender, in this case, does not discard any explicit ACKs in order to focus on the events of packet loss, duplicate ACKs, and RTO.

Students should collect statistics such as the changes in *rwnd*, *cwnd*, and the number of bytes consumed each time on receiver side in their implementations. They should also adjust attributes such as window size, traffic pattern (e.g. always send fixed length of data), and consumption style (e.g. consume a packet after a fixed small delay) to simulate various scenarios and compare throughput among those configurations. Based on the data they collected, students should then diagram and investigate the entire procedure of how the changes in those attributes impact receive window and congestion window.

IV. RESULTS

We use the PIE learning framework in an introductory networking course at undergraduate level with the objective of helping students comprehend the core techniques in the transport layer and engaging them in effective learning. The framework lets students analyze protocol essentials in detail, construct models, and evaluate their ideas by experimenting and testing with actual network traffic. We expect students to learn what went wrong and how to fix them in the procedure and to develop higher-order thinking skills [27] that can transit them from “I can follow rules to describe a procedure, but I do not know how to connect them together” to “I can explain why things happened in this way, and I can apply my ideas and compare different designs”.

TABLE I: Test Topics and Objectives

	Topics	Objective (test how well students understand…)
test 1	acknowledgement and retransmission mechanisms	how TCP detects packet loss and ensures reliability in various scenarios
test 2	connection setup and close, state transitions	1) when a state is moved to another 2) how state transition reacts to header flags 3) the changes in header fields during a complete communication 4) the information sender and receiver exchange during connection setup
test 3	flow control (fc), sliding window, receive window	1) how to adjust sliding window 2) how fc is used to pause a sender
test 4	congestion control (cc)	1) how to adjust *cwnd* according to different events 2) how cc impacts throughput

For each networking technique illustrated in our framework, we give two tests in a stepwise learning process. Table I shows the test topics and objectives. We first take a test after lecture and practice in network simulators and/or animations, depending on the topics we discussed, then, students work in the framework, and after that, we give another test on the same topic but with more advanced questions. Finally, we grade these tests and review student progress. Figure 10 shows the comparison of their grades before and after participating in the learning framework.

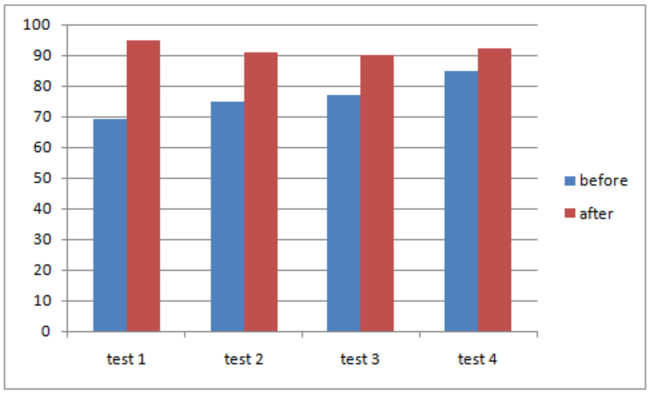


Figure 10: Student Assessment before and after Participating in each test

In error control, which deals with techniques such as sequence number, ACK, and retransmission, students gain significant improvements after learning in the framework because the techniques TCP uses to guarantee reliability are too abstract when they were discussed in classes by using diagrams or animation examples. Although existing simulators allow users to configure parameters such as packet loss rate to specify a lossy environment, how data are delivered reliably is still transparent to students. Our learning framework not only provides multiple ways for students to investigate core techniques in detail, but also creates a personalized learning environment in which they can learn things at their own pace. Similar improvement occurs in flow control, connection management, and state transition activities. In congestion control activity, students already did well in the first round because their understanding of the fundamentals is improved in the previous activities.

TABLE II: Percentage of Students who made Common Errors before and after learning in PIE

Common Errors	Before	After
ACK number does not reflect expected seq.no	*29%*	*7%*
incorrect seq.no	*12%*	*0%*
incorrect ACK number in case of packet loss	*31%*	*3%*
does not consider cumulative ACK	*22%*	*3%*
move sliding window incorrectly	*26%*	*10%*
sender overflows receiver's window	*19%*	*6%*

Table II summarizes and lists the percentage of students who makes common errors before and after their learning in PIE. It indicates that students have shown remarkable progress in all aspects. This confirms that students have gained better comprehension of those core techniques, and have developed well-organized, structured knowledge after their learning activities in the framework.

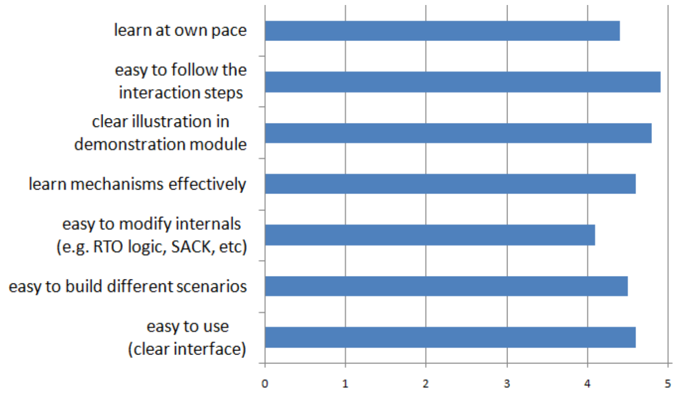


Figure 11: Student Evaluations on PIE

Students were asked to evaluate the framework from several aspects. Figure 11 shows their overall evaluation (1 means “strongly disagree”; 5 means “strongly agree”). They have identified that the interaction module is their most favorite part, which directs them step by step to complete a data transfer procedure, especially when they make mistakes the module provides hints to help them learn header details and state transition. Figure 12 lists the time they spent on the implementation part. Most students can finish the coding activity within 10 days, and based on our observation, this programming project does not cause any time conflict with other networking assignments handed out at the same time. We plan to let students implement more details in flow control and congestion control in the future.

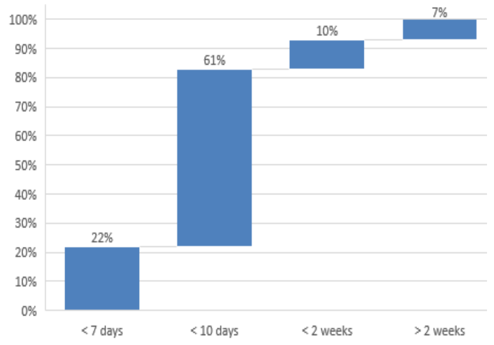


Figure 12: Time Students Spend on Mechanism Implementation

V. CONCLUSION

This paper introduces an educational framework for effective learning of TCP essentials such as error control, flow control, congestion control, connection management, and state transitions. A good understanding of these core techniques not only helps students comprehend the architecture and internal principles of network communication, but also prepares students for advanced topics and skills because these techniques have significant effect on the design of other network protocols and mechanisms. There are three modules in our framework: demonstration, interaction, and implementation. The demonstration module illustrates the details of error control by using actual network traffic and allows students to investigate how TCP detects errors and recovers data to guarantee reliability in configurable lossy environments. The interaction module focuses on the learning and practicing of packet header details and state transitions. Its self-directive GUI interacts with students through entire TCP communication procedures: from connection setup, data transfer, to connection close. In each interactive step, students need to construct a packet with appropriate values in the header fields and move the current TCP state to the next appropriate one. The interface can detect errors in student response and provide hints on how to fix them. In the implementation module, students use *socket* programming model to implement TCP flow control and congestion control. This program project will help students obtain a deep understanding of when and how to adjust transfer windows appropriately in response to various network events.

One important feature provided by our framework is that students can learn network fundamentals at their own pace. The framework uses actual network traffic in user-defined lossy environments, it allows students to introduce various error models step by step to investigate how TCP reacts to different situations. For instance, students can start from the ideal environment in which the lossy model is disabled in order to get familiar with the basic work flow. Then, various lossy models can be introduced to the receiver side only in order to focus on the details of how sender detects packet loss and when a retransmission is necessary. Next, lossy models can be activated on sender side to generate more network events, thus creating a personalized learning environment in which students can configure the framework as needed and learn things at their own pace.

According to our observation, this framework can be used not only in classrooms to demonstrate essential mechanisms in various scenarios, but also in labs or after class to provide programming projects and hands-on practices to enhance effective learning and encourage personalized learning.

REFERENCES

[1] J. Schönwälder, T. Friedman, A. Pras. Using Networks to Teach about Networks (Report on Dagstuhl Seminar #17112). *ACM SIGCOMM Computer Communication Review*. Vol 47.3, July 2017.

[2] InfiniBand Architecture Specification Volume 1, Release 1.3, InfiniBand Trade Association Std., Mar. 2015. [Online]. Available: https://infinibandta.org

[3] Supplement to InfiniBand Architecture Specification, Volume 1 Release 1.2.1 Annex A17: RoCE-v2, Sep. 2014. [Online]. Available: https://infinibandta.org

[4] Q. Liu and R. D. Russell. IBRMP: A Reliable Multicast Protocol for InfiniBand. In *Proceedings of the 2014 IEEE 22nd Annual Symposium on High-Performance Interconnects*. Aug 2014.

[5] M. den Burger and T. Kielmann. Collective Receiver-Initiated Multicast for Grid Applications. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, no. 2. Feb. 2011.

[6] E. Galip and H. Bulut. Implementing the Distributed Breadth First Search Algorithm in OMNeT++ for Teaching and Learning Purpose. *The Eurasia Proceedings of Educational & Social Sciences*. Vol 5. 2016.

[7] A. Fonseca, A. Camoes, and T. Vazao. Geographical routing implementation in ns3. *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*. Mar 2012.

[8] J. Kurose, and K. Ross. *Computer Networking: A Top-Down Approach (7th edition)*. Pearson. May 2016.

[9] M. Holiday. Animation of computer networking concepts. *Journal on Educational Resources in Computing*. Jun 2003.

[10] R. Chang. Teaching computer networking with the help of personal computer networks. *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. Jun 2004.

[11] OMNeT++ Simulator. [Online]. Available: https://omnetpp.org

[12] NS-3 Network Simulator. [Online]. Available: https://www.nsnam.org

[13] Q. Liu, R. Russell, and E. Gran. Improvements to the InfiniBand Congestion Control Mechanism. *2016 IEEE 24th Annual Symposium on High-Performance Interconnects*. Aug 2016.

[14] J. Jiang, Y. Li, S. Hong, A. Xu, and K. Wang. A Time-sensitive Networking (TSN) Simulation Model Based on OMNET++. *2018 IEEE International Conference on Mechatronics and Automation (ICMA)*. Aug 2018.

[15] N. Baldo, M. Requena-Esteso, J. Nunez-Martinez, M. Portoles-Comeras, J. Nin-Guerrero, P. Dini, and J. Mangues-Bafalluy. Validation of the IEEE 802.11 MAC model in the ns3 simulator using the EXTREME testbed. *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. Mar 2010.

[16] G. Piro, N. Baldo, and M. Miozzo. An LTE module for the ns-3 network simulator. *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*. Mar 2011.

[17] GNS3 Simulator. [Online]. Available: https://www.gns3.com

[18] R. Mohtasin, et. al. Development of a virtualized networking lab using GNS3 and VMWARE workstation. In *2016 International Conference on Wireless Communications, Signal Processing and Networking*. Mar 2016.

[19] Cisco Packet Tracer. [Online]. Available: https://www.netacad.com/courses/packet-tracer

[20] D. Feinberg. Teaching Simplified Network Protocols. In *Proceedings of the 41st ACM technical symposium on Computer science education*. Mar 2010.

[21] OPNET Simulator. [Online]. Available: https://www.riverbed.com/products/steelcentral/opnet.html

[22] B. Momeni, and M. Kharrazi. Improving a computer networks course using the Partov simulation engine. *IEEE Transactions on Education*. Vol 55.3. Aug 2012.

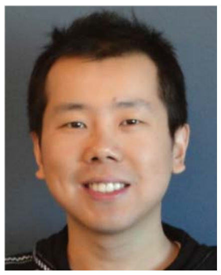
[23] J. Pullen. Teaching network protocol concepts in an open-source simulation environment. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. Jul 2018.

[24] W. Zhu. Hands-on network programming projects in the cloud. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. Mar 2015.

[25] M. El-Kharashi, G. Darling, B. Marykuca, and G. C. Shoja. Understanding and implementing computer network protocols through a lab project. *IEEE Transactions on Education*. Vol 45.3. Aug 2002.

[26] K. Lee, J. Kim, and S. Moon. An educational networking framework for full layer implementation and testing. In *Proceedings of the 2014 ACM conference on SIGCOMM*. Aug 2014.

[27] L. W. Anderson, et. al. *A taxonomy for learning and teaching and assessing: A revision of Bloom's taxonomy of educational objectives*. Addison Wesley Longman, Inc, 2001.



Dr. Qian Liu is an Assistant Professor with Mathematics and Computer Science department at Rhode Island College, Providence, USA. Dr. Liu received the Ph.D. degree in Computer Science from the University of New Hampshire in 2016. His research interests include computer networks, operating system, simulation, and learning technologies.